Course: The XPath Language

Pierre Genevès
CNRS

University of Grenoble, 2012–2013
Search, selection and extraction of information from XML documents are essential for any kind of XML processing.

XPath is the W3C standard language for expressing traversal and navigation in XML trees.
XPath Introduction

- A common syntax and semantics for many web languages
- A W3C recommendation ([www.w3.org/TR(xpath)](http://www.w3.org/TR/xpath))
- Compact syntax, not in XML, for use within XML attributes
- A language for expressing paths
- XPath operates on the logical (tree) structure of XML documents, not on their syntax
XPath Expressions

• XPath provides a powerful mechanism for navigating in XML trees: the location path

• A location path is a sequence of location steps separated by '/':

child :: chapter / descendant :: section / child :: para
Evaluating a *location path*

- Starting from a context node, a *location path* returns a *node-set*.
- Each node of this *node-set* becomes in turn the context node for evaluating the next *step*.
Evaluating a *location path*

- Starting from a context node, a *location path* returns a *node-set*
- Each node of this *node-set* becomes in turn the context node for evaluating the next *step*
Evaluating a *location path*

- Starting from a context node, a *location path* returns a *node-set*
- Each node of this *node-set* becomes in turn the context node for evaluating the next *step*
Evaluating a location path

- Starting from a context node, a location path returns a node-set.
- Each node of this node-set becomes in turn the context node for evaluating the next step.

```
descendant::c/child::E
```
Evaluating a location path

- Starting from a context node, a location path returns a node-set.
- Each node of this node-set becomes in turn the context node for evaluating the next step.

```
descendant::*C/child::*E/child::*F
```
Evaluation Context

- Every XPath expression is evaluated with respect to a context that includes:
  - the context node
  - 2 integers > 0 obtained from the evaluation of the last step:
    - context size: the number of nodes in the node-set
    - context position: the index of the context node in the node-set
  - a set of variable bindings (expressed in the host language)

- Navigation “propagates” the context: evaluation of a step yields a new context state

- Remark: a location path starting with ‘/’ indicates that the initial context is set to the root of the document, such a location path is called “absolute”
Zoom on location steps

- A each navigation step, nodes can be filtered using **qualifiers**
- General syntax of a **location step**:

  \[ \text{axis}::\text{nodetest}[\text{qualifier}][\text{qualifier}] \]

- A **location step** is composed of 3 parts:
  1. an **axis**: specify the relation between the context node and returned nodes
  2. a **nodetest**: type and name of returned nodes
  3. optional **qualifiers** that further filter nodes

- Qualifiers are applied one after the other, once the selection is performed by the **axis** and **nodetest**
- A qualifier returns a **node-set** that is filtered by the next qualifier
- Example: \text{child}::\text{section}[\text{child}::\text{para}][\text{child}::\text{b}]
Axes

- Indicates where in the tree (with respect to the context node) selected nodes must be searched
- XPath defines 13 *axes* allowing navigation, including:

![](image)

- 5 *axes* define a partition of tree nodes
• Each *axis* has a direction: forward or backward (w.r.t *document ordering*)

• Other axes:
  • ancestor-or-self, descendant-or-self
  • attribute: selects attributes of the context node (element)
  • namespace: selects namespace nodes of the context node
The nodetest of a location step indicates which nodes must be chosen on the considered axis.

A nodetest filters nodes, e.g.:

<table>
<thead>
<tr>
<th>Test</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>node()</td>
<td>let any node pass</td>
</tr>
<tr>
<td>text()</td>
<td>preserve only text nodes</td>
</tr>
<tr>
<td>comment()</td>
<td>preserve only comment nodes</td>
</tr>
<tr>
<td>name</td>
<td>preserve only elements/attributes with tag “name”</td>
</tr>
<tr>
<td>*</td>
<td>preserve arbitrary elements/attributes</td>
</tr>
</tbody>
</table>

Remarks:

1. $\text{path/child::*} \subseteq \text{path/child::node()}$
2. $\text{path/attribute::node()} \nsubseteq \text{path/child::node()}$
• A **qualifier** filters a *node-set* depending on the *axis* and returns a new *node-set*

• A **qualifier** is a boolean expression evaluated depending on the *context*:
  • context node
  • *context size*: number of nodes in the *node-set*
  • *context position*: index of the context node in the *node-set*, in the order of the document (or in reverse document order for *backward* axes)

• Each node of a *node-set* is kept only if the evaluation of the **qualifier** for this node returns *true*

• Examples:
  • `following-sibling::para[position()=last()]`
  • `child::para[position() mod 2 = 1]`
Value Comparisons

• **Qualifiers** may include comparisons:

\[
\text{path}[\text{path}_1 \ e q \ \text{path}_2] \quad \text{eq} \in \{=, \neq, <, >, <=, >=\}
\]

• **Existential semantics:**

\[
\text{node-set}_1 \ e q \ \text{node-set}_2 \quad \text{iff} \quad \exists n_1 \in \text{node-set}_1, \exists n_2 \in \text{node-set}_2 \mid \text{string-value}(n_1) \ e q \ \text{string-value}(n_2)
\]
Value Comparisons

- **Qualifiers** may include comparisons:
  
  \[ path[path_1 \text{ eq } path_2] \quad \text{eq} \in \{=, \neq, <, >, \leq, \geq \} \]

- **Existential semantics:**
  
  \[ \text{node-set}_1 \text{ eq } \text{node-set}_2 \quad \text{iff} \]
  
  \[ \exists n_1 \in \text{node-set}_1, \exists n_2 \in \text{node-set}_2 \mid \text{string-value}(n_1) \text{ eq } \text{string-value}(n_2) \]

- **string-value(n):** concatenation of all descendant text nodes in *document order*

- **Example:** descendant::chapter[child::section="Conclusion"]
  
  → all "chapter" nodes whose at least one "section" child has string-value "Conclusion".
Value Comparisons

- **Qualifiers** may include comparisons:

  \[ path[p_1 \text{ eq } p_2] \quad \text{eq} \in \{=, !=, <, >, <=, >=\} \]

- **Existential semantics:**

  \[ n_1 \in n_1 \text{ eq } n_2 \iff \exists n_1 \in n_1, \exists n_2 \in n_2 \mid \text{string-value}(n_1) \text{ eq } \text{string-value}(n_2) \]

- **string-value(n):** concatenation of all descendant text nodes in *document order*

- **Example:** \( \text{descendant::chapter[child::section="Conclusion"]} \)
  \[ \rightarrow \text{all "chapter" nodes whose at least one "section" child has string-value "Conclusion".} \]

- **Comparisons may involve (implicit) type casting (ex: a[b>7] )**
A general XPath expression is a location path, or a union of location paths separated by ‘|’.

Qualifiers may include boolean expressions:
\[ \text{path}[(\text{path } \text{eq } \text{path}) \text{ or } (\text{qualifier and not(qualifier))}] \]

An XPath expression may include variables (notation: $x$)
- variables are bound by the host language (i.e. they are constants 😊)
- they are part of the evaluation context
Observation on Data Value Comparisons

- Assume variable $x$ is bound to a *node-set*
- What do you think of the following XPath expressions $e_1$ and $e_2$?

\[
\begin{align*}
\text{\$x = "foo"} & \quad \text{not(\$x != "foo")} \\
\text{\(e_1\)} & \quad \text{\(e_2\)}
\end{align*}
\]
Observation on Data Value Comparisons

• Assume variable $x$ is bound to a node-set

• What do you think of the following XPath expressions $e_1$ and $e_2$?

$e_1$ is different from $e_2$:

→ $e_1$ is true iff there exists a node in $x$ which has string-value foo;

→ $e_2$ is true iff all nodes in $x$ have string string-value foo.
Observation on Data Value Comparisons

- Assume variable $x$ is bound to a node-set

- What do you think of the following XPath expressions $e_1$ and $e_2$?
  \[
  \begin{align*}
  e_1 & : \not (x \neq \text{foo}) \\
  e_2 & : \not (x \neq \text{foo})
  \end{align*}
  \]

- $e_1$ is different from $e_2$:
  \[\rightarrow \text{ e}_1 \text{ is true iff there exists a node in } x \text{ which has string-value } \text{foo};\]
  \[\rightarrow \text{ e}_2 \text{ is true iff all nodes in } x \text{ have string string-value } \text{foo}.\]

- Owing to negation and comparison defined by existential quantification, we can formulate universal quantification...
Observation on Data Value Comparisons

• Assume variable $x$ is bound to a node-set

• What do you think of the following XPath expressions $e_1$ and $e_2$?

$$
\begin{align*}
&\text{\texttt{\$x=\textquote{\textit{foo}}} } & \text{\texttt{not($x\neq\textquote{\textit{foo}}$)}} \\
&\text{\texttt{$e_1$}} & \text{\texttt{$e_2$}} 
\end{align*}
$$

• $e_1$ is different from $e_2$:

$\rightarrow$ $e_1$ is true iff there exists a node in $x$ which has string-value foo;

$\rightarrow$ $e_2$ is true iff all nodes in $x$ have string string-value foo.

• Owing to negation and comparison defined by existential quantification, we can formulate universal quantification...

• “chapter” nodes whose all children “section” are empty? 

\footnote{have an empty string-value}
Observation on Data Value Comparisons

• Assume variable $x$ is bound to a node-set

• What do you think of the following XPath expressions $e_1$ and $e_2$?

\[
\begin{align*}
&\underbrace{x="foo"}_{e_1} & \underbrace{\neg(x\neq"foo")}_{e_2}
\end{align*}
\]

• $e_1$ is different from $e_2$:
  → $e_1$ is true iff there exists a node in $x$ which has string-value $foo$;
  → $e_2$ is true iff all nodes in $x$ have string string-value $foo$.

• Owing to negation and comparison defined by existential quantification, we can formulate universal quantification...

  • “chapter” nodes whose all children “section” are empty\(^1\)?
  → descendant::chapter[not(child::section!’="’)]

\(^1\)have an empty string-value
Basic Functions

- **Node-sets** are not the only types of XPath expressions: there are *boolean*, *numerical* and *string* expressions too.

- Every XPath implementation must provide at least a list of basic functions called *Core Function Library* (c.f. appendix).

- Examples:
  - `last()`: a number, the *context size*
  - `position()`: a number, the *context position*
  - `count(node-set)`: number of nodes in the *node-set*
  - `concat(string, string, string*)`: concatenate several strings
  - `contains(str1, str2)`: boolean, true if *str1* contains *str2*
  - ...

- Any XPath expression can be used within a *qualifier*, for instance:

  ```xml
  descendant::recipe[count(descendant::ingredients)<5 and contains(child::title, "cake")]
  ```
Abbreviated Syntax

- child:: is the default axis, it can be omitted
- @ is a shorthand for attribute::
- // is a shorthand for /descendant-or-self::node()/
- . is a shorthand for self::node()
- .. is a shorthand for parent::node()
- [4] is a shorthand for [position()=4]

<table>
<thead>
<tr>
<th>Example</th>
<th>Expanded Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>book/section</td>
<td>child::book/child::section</td>
</tr>
<tr>
<td>p[@id=&quot;bla&quot;]</td>
<td>child::p[attribute::id=&quot;bla&quot;]</td>
</tr>
<tr>
<td>.//p</td>
<td>self::node()/descendant-or-self::node()/child::p</td>
</tr>
<tr>
<td>../title</td>
<td>parent::node()/child::title</td>
</tr>
<tr>
<td>p[3]</td>
<td>child::p[position()=3]</td>
</tr>
</tbody>
</table>
What do you think of the following XPath expressions $e_1$ et $e_2$?

$e_1$: self::title

$e_2$: parent::node()/child::title
Can we rewrite the XPath expression `following::p` without the axis `following`?
XPath: A Core Component for XML Technologies

• XPath is used in:
  • **XSLT**: selection of document parts to be transformed
  • **XQuery**: XPath is the (main) subset of the query language
  • **XPointer**: identification of XML fragments
  • **XLink**: definition of hypertext links
  • **XML Schema**: expressing the tree region in which unicity is guaranteed
  • **XForms**: expressing dependencies (data bindings)
  • ...

• Often, it is even the **essential** component
Many different ways to express navigation to the same nodes

Two XPath expressions might share the same semantics even if they differ syntactically (and operationally!)

$\text{child}::\text{a}[\text{child}::\text{b}] / \text{following-sibling}::\text{c}$

$\text{child}::\text{c}[\text{preceding-sibling}::\text{a}[\text{child}::\text{b}]]$

Determining query equivalence is crucial (e.g. optimization)

---

$^2$The semantics of an XPath expression is to be understood as the final set of nodes resulting from the evaluation of the expression.
• What about the following expressions?

\texttt{descendant::d[parent::b]/following-sibling::a}

\texttt{ancestor-or-self::*/descendant-or-self::b/a[preceding-sibling::d]}

• Question for next time(s): how would you write a program that checks whether two XPath expressions are equivalent (i.e. return the same set of nodes when applied from the same context in any tree)?
XPath Core Function Library
Functions over node-sets

- `last()`: a number, the context size
- `position()`: a number, the context position
- `count(node-set)`: number of nodes in the node-set
- `id(object)`: selects elements by their unique ID
- `local-name(node-set)`: returns the local part of the expanded-name of the node in the argument node-set that is first in document order.
- `namespace-uri(node-set)`: returns the namespace URI of the expanded-name of the node in the argument node-set that is first in document order
- `name(node-set)`: returns a string containing the whole name of the node in the argument node-set that is first in document order
String Functions

- `string(object)`: convert `object` to a string
- `concat(string, string, string*)`: concatenate several strings
- `start-with(string1, string2)`: boolean, true if `string1` starts with `string2`
- `contains(str1, str2)`: boolean, true if `str1` contains `str2`
- `substring-before(string1, string2)`: the substring of `string1` before the first occurrence of `string2`
- `substring-after(string1, string2)`: the substring of `string1` after the first occurrence of `string2`
- `substring(string, number1, number2)`: the substring of `string` that starts at position `number1` and whose length is `number2`
- `string-length(string)`: number of characters in `string`
- `normalize-space(string)`: remove beginning, ending and double spaces
- `translate(s1, s2, s3)`: replace in `s1` each char of `s2` by the char of same position in `s3`

Example: `translate("bar","abc","ABC")` returns `BAr`
Boolean Functions

- `boolean(object)`: convert `object` into boolean, returns true if non zero number, non empty `node-set`, string with non zero length
- `not(boolean)`: negation of `boolean`
- `true()`
- `false()`
- `lang(string)`: the language (attribute `xml:lang`) of context node is the same or a sublanguage of `string`
Arithmetic Functions

- `number(object)`: convert `object` into a number
- `sum(node-set)`: sum of the (type casted) number representation of each node in the `node-set`
- `floor(number)`: greatest integer less or equal to `number`
- `ceiling(number)`: smallest integer greater than or equal to `number`
- `round(number)`: the closest integer of `number`
Operator Precedence

1. $\leq$, $<$, $\geq$, $>$
2. $=$, $\neq$
3. and
4. or